LPI exam 301 prep, Topic 302: Installation and development

Senior Level Linux Professional (LPIC-3)

Skill Level: Intermediate

Sean Walberg (sean@ertw.com)
Network engineer
Freelance

04 Dec 2007

In this tutorial, Sean Walberg helps you prepare to take the Linux Professional Institute® Senior Level Linux Professional (LPIC-3) exam. In this second in a series of six tutorials, Sean walks you through installing and configuring a Lightweight Directory Access Protocol (LDAP) server, and writing some Perl scripts to access the data. By the end of this tutorial, you'll know about LDAP server installation, configuration, and programming.

Section 1. Before you start

Learn what these tutorials can teach you and how you can get the most from them.

About this series

The Linux Professional Institute (LPI) certifies Linux system administrators at three levels: *junior level* (also called "certification level 1"), *advanced level* (also called "certification level 2"), and *senior level* (also called "certification level 3"). To attain certification level 1, you must pass exams 101 and 102. To attain certification level 2, you must pass exams 201 and 202. To attain certification level 3, you must have an active advanced-level certification and pass exam 301 ("core"). You may also pass additional specialty exams at the senior level.

developerWorks offers tutorials to help you prepare for the five junior, advanced, and senior certification exams. Each exam covers several topics, and each topic has a corresponding self-study tutorial on developerWorks. Table 1 lists the six topics and

corresponding developerWorks tutorials for LPI exam 301.

Table 1. LPI exam 301: Tutorials and topics

LPI exam 301 topic	developerWorks tutorial	Tutorial summary
Topic 301	LPI exam 301 prep: Concepts, architecture, and design	Learn about LDAP concepts and architecture, how to design and implement an LDAP directory, and about schemas.
Topic 302	LPI exam 301 prep: Installation and development	(This tutorial) Learn how to install, configure, and use the OpenLDAP software. See the detailed objectives.
Topic 303	LPI exam 301 prep: Configuration	Coming soon.
Topic 304	LPI exam 301 prep: Usage	Coming soon.
Topic 305	LPI exam 301 prep: Integration and migration	Coming soon.
Topic 306	LPI exam 301 prep: Capacity planning	Coming soon.

To pass exam 301 (and attain certification level 3), you should:

- Have several years of experience in installing and maintaining Linux on a number of computers for various purposes
- Have integration experience with diverse technologies and operating systems
- Have professional experience as, or training to be, an enterprise-level Linux professional (including having experience as a part of another role)
- Know advanced and enterprise levels of Linux administration including installation, management, security, troubleshooting, and maintenance.
- Be able to use open source tools to measure capacity planning and troubleshoot resource problems
- Have professional experience using LDAP to integrate with UNIX® services and Microsoft® Windows® services, including Samba, Pluggable Authentication Modules (PAM), e-mail, and Active Directory
- Be able to plan, architect, design, build, and implement a full environment using Samba and LDAP as well as measure the capacity planning and security of the services
- Be able create scripts in Bash or Perl or have knowledge of at least one system programming language (such as C)

The Linux Professional Institute doesn't endorse any third-party exam preparation

material or techniques in particular.

About this tutorial

Welcome to "Installation and development," the second of six tutorials designed to prepare you for LPI exam 301. In this tutorial, you learn about LDAP server installation and configuration, and how to use Perl to access your new LDAP server.

This tutorial is organized according to the LPI objectives for this topic. Very roughly, expect more questions on the exam for objectives with higher weights.

Objectives

Table 2 shows the detailed objectives for this tutorial.

Table 2. Installation and development: Exam objectives covered in this tutorial

LPI exam objective	Objective weight	Objective summary
302.1 Compiling and installing OpenLDAP	3	Compile and install OpenLDAP from source and from packages
302.2 Developing for LDAP with Perl/C++	1	Write basic Perl scripts to interact with an LDAP directory

Prerequisites

To get the most from this tutorial, you should have advanced knowledge of Linux and a working Linux system on which to practice the commands covered.

If your fundamental Linux skills are a bit rusty, you may want to first review the tutorials for the LPIC-1 and LPIC-2 exams.

Different versions of a program may format output differently, so your results may not look exactly like the listings and figures in this tutorial.

System requirements

To follow along with the examples in these tutorials, you'll need a Linux workstation with the OpenLDAP package and support for PAM. Most modern distributions meet these requirements.

Section 2. Compiling and installing OpenLDAP

This section covers material for topic 302.1 for the Senior Level Linux Professional (LPIC-3) exam 301. This topic has a weight of 3.

In this section, learn how to:

- Compile and configure OpenLDAP from source
- Understand OpenLDAP backend databases
- Manage OpenLDAP daemons
- Troubleshoot errors during installation

OpenLDAP is an open source application that implements an LDAP server and associated tools. Because it's open source, you can download the source code free of charge. The OpenLDAP project doesn't distribute binaries directly, but most major distributions package it themselves. In this tutorial, you learn how to install OpenLDAP from both source and packages.

Compiling from source

The first order of business is to download the latest version of OpenLDAP from the project site (see the Resources section for a download link). The project generally has two active versions available: One is a stable version, and the other is a test version. This tutorial was written using the stable versions 2.3.30 and 2.3.38. If you're following along, some of the directory names may be different depending on which version you have.

To extract the source code from the downloaded tarball, enter `tar -xzf openldap-stable-20070831.tgz`. This decompresses and untars the downloaded file into a directory. Change into the new directory with `cd openldap-2.3.38` (substituting your version of OpenLDAP as appropriate).

At this point, you're in the source directory. You must now configure the build environment for your system and then build the software. OpenLDAP uses a script called `configure` that performs these actions. Type `./configure --help` to see all the options available to you. Some define where the files are installed to (such as `--prefix`); others define the OpenLDAP features you wish to build. Listing 1 lists the features and their defaults.

Listing 1. Configuration options relating to OpenLDAP features

```
SLAPD (Standalone LDAP Daemon) Options:
  --enable-slapd          enable building slapd [yes]
    --enable-aci          enable per-object ACIs (experimental) [no]
    --enable-cleartext    enable cleartext passwords [yes]
```

```
    --enable-crypt        enable crypt(3) passwords [no]
    --enable-lmpasswd     enable LAN Manager passwords [no]
    --enable-spasswd      enable (Cyrus) SASL password verification [no]
    --enable-modules      enable dynamic module support [no]
    --enable-rewrite      enable DN rewriting in back-ldap and rwm overlay [auto]
    --enable-rlookups     enable reverse lookups of client hostnames [no]
    --enable-slapi        enable SLAPI support (experimental) [no]
    --enable-slp          enable SLPv2 support [no]
    --enable-wrappers     enable tcp wrapper support [no]

SLAPD Backend Options:
    --enable-backends     enable all available backends no|yes|mod
    --enable-bdb          enable Berkeley DB backend no|yes|mod [yes]
    --enable-dnssrv       enable dnssrv backend no|yes|mod [no]
    --enable-hdb          enable Hierarchical DB backend no|yes|mod [yes]
    --enable-ldap         enable ldap backend no|yes|mod [no]
    --enable-ldbm         enable ldbm backend no|yes|mod [no]
      --enable-ldbm-api   use LDBM API auto|berkeley|bcompat|mdbm|gdbm [auto]
      --enable-ldbm-type  use LDBM type auto|btree|hash [auto]
    --enable-meta         enable metadirectory backend no|yes|mod [no]
    --enable-monitor      enable monitor backend no|yes|mod [yes]
    --enable-null         enable null backend no|yes|mod [no]
    --enable-passwd       enable passwd backend no|yes|mod [no]
    --enable-perl         enable perl backend no|yes|mod [no]
    --enable-relay        enable relay backend no|yes|mod [yes]
    --enable-shell        enable shell backend no|yes|mod [no]
    --enable-sql          enable sql backend no|yes|mod [no]

SLAPD Overlay Options:
    --enable-overlays     enable all available overlays no|yes|mod
    --enable-accesslog    In-Directory Access Logging overlay no|yes|mod [no]
    --enable-auditlog     Audit Logging overlay no|yes|mod [no]
    --enable-denyop       Deny Operation overlay no|yes|mod [no]
    --enable-dyngroup     Dynamic Group overlay no|yes|mod [no]
    --enable-dynlist      Dynamic List overlay no|yes|mod [no]
    --enable-lastmod      Last Modification overlay no|yes|mod [no]
    --enable-ppolicy      Password Policy overlay no|yes|mod [no]
    --enable-proxycache   Proxy Cache overlay no|yes|mod [no]
    --enable-refint       Referential Integrity overlay no|yes|mod [no]
    --enable-retcode      Return Code testing overlay no|yes|mod [no]
    --enable-rwm          Rewrite/Remap overlay no|yes|mod [no]
    --enable-syncprov     Syncrepl Provider overlay no|yes|mod [yes]
    --enable-translucent  Translucent Proxy overlay no|yes|mod [no]
    --enable-unique       Attribute Uniqueness overlay no|yes|mod [no]
    --enable-valsort      Value Sorting overlay no|yes|mod [no]

SLURPD (Replication Daemon) Options:
  --enable-slurpd         enable building slurpd [auto]

Optional Packages:
  --with-PACKAGE[=ARG]    use PACKAGE [ARG=yes]
  --without-PACKAGE       do not use PACKAGE (same as --with-PACKAGE=no)
  --with-subdir=DIR       change default subdirectory used for installs
  --with-cyrus-sasl       with Cyrus SASL support [auto]
  --with-fetch            with fetch(3) URL support [auto]
  --with-threads          with threads [auto]
  --with-tls              with TLS/SSL support [auto]
  --with-yielding-select  with implicitly yielding select [auto]
  --with-odbc             with specific ODBC support iodbc|unixodbc|auto [auto]

  --with-gnu-ld           assume the C compiler uses GNU ld [default=no]
  --with-pic              try to use only PIC/non-PIC objects [default=use
                          both]
  --with-tags[=TAGS]
                          include additional configurations [automatic]
```

In Listing 1, you can see that many features are disabled by default, such as metadirectories and modules. In addition, many options are marked as "auto," which turns on features if the proper libraries are present on your system. Instead of relying on this automatic behavior, it's best to make a list of the required features and

enable them. If you're missing any libraries, you'll get an error to this effect at compile time, rather than some time later.

Some configuration options can be passed either `no`, `yes`, or `mod`. `no` disables the option, `yes` causes the option to be statically linked to the final binary, and `mod` builds the option as a separate shared library. Shared libraries are loaded into the server at runtime (see "Server parameters (global)" below). By default, the modules are statically linked; that is, they are part of the binary and inseparable. If you wish to use dynamic modules, you will also need the `--enable-modules` option. The benefits of dynamic modules are that you can test various options without bloating your binary, and you can package the modules separately.

Listing 2 shows a configuration line, based on the configuration that ships in Fedora 7, that enables many helpful features. For the most part, the chosen options will enable features that will be required in later tutorials, such as `--enable-slurpd`and `--enable-multimaster` for replication, and `--enable-meta` for meta-directories. Other options enable various backends, such as ldab, bdb, null, and monitor.

Listing 2. A sample build configuration

```
./configure --enable-plugins --enable-modules --enable-slapd --enable-slurpd \
   --enable-multimaster --enable-bdb --enable-hdb --enable-ldap --enable-ldbm \
   --enable-ldbm-api=berkeley --enable-meta --enable-monitor --enable-null \
   --enable-shell --enable-sql=mod --disable-perl \
   --with-kerberos=k5only --enable-overlays=mod --prefix=/tmp/openldap
```

Listing 2 enables plug-ins and multiple backends, including Structured Query Language (SQL) based backends and Berkeley Database files. Backends are OpenLDAP's way of storing and retrieving data, and are examined in more detail under "Backends and databases," and in later tutorials.

Listing 2 also builds both the stand-alone daemon `slapd` and the replication daemon `slurpd`. Overlays, which allow easier customization of the backend data, are also enabled for testing. Because this is a test setup, the installation prefix has been changed to `/tmp/openldap`, so the resulting binaries end up in `/tmp/openldap/libexec`.

When you execute the `configure` script, it checks for the necessary libraries and then generates the build environment. If `configure` completes successfully, compile OpenLDAP with `make depend; make`.

After the code has compiled, you can install OpenLDAP with `make install`. This copies all the binaries, manpages, and libraries to their place in `/tmp/openldap`.

Installing from packages

If you were daunted by the previous section on compiling from source, you aren't alone. Compiling from source is time consuming and can be aggravating if you don't have the proper development libraries available. If your C development experience is limited or nonexistent, then you'll likely have trouble interpreting any build errors.

Fortunately, most distributions package OpenLDAP as a set of binaries with a preset configuration. Usually these binaries have all the features you'll ever need.

RPM-based distributions

Fedora and CentOS use the `yum` tool to install RedHat packages (RPMs) from repositories. To find out which packages are available, use the `yum list` command, passing an optional regular expression that filters the list of packages returned. Listing 3 shows a search for all packages containing the term `openldap`.

Listing 3. Determining which packages are available through yum

```
# yum list \*openldap\*
Loading "installonlyn" plugin
Setting up repositories
Reading repository metadata in from local files
Installed Packages
openldap.i386                           2.3.30-2.fc6           installed
openldap-clients.i386                   2.3.30-2.fc6           installed
openldap-devel.i386                     2.3.30-2.fc6           installed
openldap-servers.i386                   2.3.30-2.fc6           installed
openldap-servers-sql.i386               2.3.30-2.fc6           installed
Available Packages
compat-openldap.i386                    2.3.30_2.229-2.fc6    updates
```

In a large application such as OpenLDAP, the client and server tools are often split into two separate packages. In addition, you may find some compatibility libraries (to ensure applications linked against much older versions of the software still work). To install a package, use `yum install` with the name of the package, such as `yum install openldap-clients openldap-servers`; this downloads and installs both the client and server packages, along with any needed dependencies.

For Red Hat Enterprise Linux, the command to search packages for `openldap` is `up2date --showall | grep openldap`. To install a package, supply the package names as arguments to `up2date`, such as `up2date openldap-clients openldap-servers`.

To make sure the OpenLDAP server starts on boot, use `chkconfig ldap on`.

Debian-based distributions

Debian-based distributions, such as Ubuntu, use the Advanced Packaging (APT) tools to install packages. First, to search for OpenLDAP packages, use `apt-cache search openldap`, as shown in Listing 4.

Listing 4. Listing the available OpenLDAP packages in Ubuntu Linux

```
notroot@ubuntu:~$ apt-cache search openldap
libldap2 - OpenLDAP libraries
libldap2-dev - OpenLDAP development libraries
python-ldap - A LDAP interface module for Python. [dummy package]
python-ldap-doc - Documentation for the Python LDAP interface module
python2.4-ldap - A LDAP interface module for Python 2.4
ldap-utils - OpenLDAP utilities
```

```
libldap-2.2-7 - OpenLDAP libraries
slapd - OpenLDAP server (slapd)
```

Listing 4 shows several packages available. The `slapd` package provides the server, and any dependencies will be resolved at install time. Run `sudo apt-get install slapd` to install the server. You may also include the `ldap-utils` package, which contains the command-line clients.

Configuring the software

Once you've installed OpenLDAP, you must configure it. For testing purposes, you need to specify only a few things; but for the real world (and the LPIC 3 exam), you must be well acquainted with the various options.

Two configuration files govern the behavior of OpenLDAP; both are in /etc/openldap/ by default. The first is ldap.conf, which controls the global behavior of LDAP clients. The configuration file for all LDAP servers is called slapd.conf. Despite the name, slapd.conf also has the configuration for `slurpd`, the replication daemon. The focus of this article is on slapd.conf, specifically pertaining to the `slapd` daemon.

slapd.conf has an easy format: a single keyword followed by one or more arguments, subject to the following conditions:

- The keyword must start at column 0—that is, no spaces may exist in front of it.
- If an argument has spaces in it, the argument must be quoted with double quotes ("").
- If a line begins with a space, it's considered a continuation of the previous line.
- Keywords aren't case sensitive, but the arguments may be, depending on which keyword is used.

As with most UNIX® tools, the hash symbol (#) denotes a comment. Anything after the hash is ignored.

slapd.conf is divided into two sections: global options and backend database options. Although this ordering isn't enforced, you must be careful where you place your directives, because some directives alter the context in which subsequent directives are processed. For instance, if no `backend` or `database` keywords have been encountered, an option is considered global. Once a `database` directive is read, all further options apply to that database. This continues until another `database` directive is read, at which point the next commands apply to the new database.

Some of the global options will be covered in later tutorials in this 301 series, such as those dealing with access controls and replication. A description of the commonly

used configuration directives follows.

Server parameters (global)

Several parameters limit the work that the `slapd` process can do, which prevents resource starvation. `conn_max_pending` accepts an integer that dictates how many anonymous requests can be pending at any given time. You'll learn about binding to the LDAP server in a later tutorial in this 301 series; simply put, you can make requests from the server by logging in as a user (an authenticated session) or without any credentials (anonymous session). Requests beyond the `conn_max_pending` limit are dropped by the server. Similarly, `conn_max_pending_auth` is the same as `conn_max_pending` but refers to authenticated sessions.

The `idletimeout` parameter (specified in seconds) tells `slapd` how long idle clients can be held before they should be disconnected. If this number is 0, no disconnections happen.

The `sizelimit` parameter limits the number of search results that can come back from a single query, and `timelimit` limits how long the server spends searching. These two parameters can take either an integer, the keyword `unlimited`, or more complex hard and soft limits. This would allow you to set a default (soft) timeout or result-set size; but if a client requests a larger number of rows or a longer timeout, it can be accommodated up to the hard limit. For example, `sizelimit sizesoft=400 size.hard=1000` specifies that by default, 400 rows are returned. Clients can request that this limit be increased up to 1,000. This format can be applied to groups of users so that some people or applications can perform large searches, and others can perform only small searches

When a client performs a search on the tree, it usually specifies a node (called the *search base*, or *base*) from which the search should start—the Distinguished Names (DNs) of all the results have the search base in them. This allows for faster searching (because fewer nodes need to be searched) and easier client implementation (because searching only part of a tree is a simple but effective filter). If the client doesn't specify a base, the value of `defaultsearchbase` is used. This is a good parameter to set to avoid surprises with misconfigured clients down the road. Depending on the layout of your LDAP tree, you may wish to use either your users container or the root of the tree. (Trees and distinguished names are covered in the previous tutorial.)

Three commands govern various features supported by your server, such as legacy support and security requirements by clients. These commands are `allow`, `disallow`, and `require`. Each command takes a series of whitespace keywords that enable, disable, or require a feature. The keywords are shown in Table 3.

Table 3. Keywords used with allow, disallow, and require

Command(s)	Keyword	Description	Default
`allow`	`bind_v2`	If set, allows legacy LDAPv2 clients to connect. The	Disallowed

		OpenLDAP documentation repeatedly points out that OpenLDAP doesn't truly support LDAPv2, so some requests may result in unexpected behavior.	
`allow`	`bind_anon_cred`	Allows a client to bind with a password but no DN. If this option is allowed, then the client is allowed as an anonymous bind.	Disallowed
`allow`	`bind_anon_dn`	Allows a client to bind with a DN but no password, usually because the client is misconfigured. If this option is allowed, then the client is allowed as an anonymous bind.	Disallowed
`allow, disallow`	`update_anon`	Allows an anonymous bind, which happens when a client connects to the LDAP server with no DN or password.	Allowed
`disallow`	`bind_simple`	Allows simple (unencrypted user names and passwords) authentication as opposed to a stronger method such as Simple Authentication and Security Layer (SASL).	Allowed
`require`	`bind`	Requires that the client bind to the directory with the **bind** operation before doing any other operations.	Not required
`require`	`LDAPv3`	Determines whether LDAPv3 is required. Note that this can conflict with `allow bind_v2`.	Not required
`require`	`authc`	Requires authentication, as	Not required

		opposed to an anonymous bind.	
`require`	`SASL`	Requires that a SASL method be used to connect to the server	Not required
`require`	`strong`	Requires that a strong method of authentication be used. This can be either SASL or simple authentication over a protected method.	Not required
`require`	`none`	This option clears out all the requirements, usually if you're relaxing the requirements for a certain database by using this command in the database section of slapd.conf. If you wish to change the requirements for the database (as opposed to just clearing the list), you must use `none` before adding your new requirements, even if they have been required in the global section.	Not applicable

Even though certain types of login may be allowed by some commands from Table 3, the connections are still subject to access controls. For example, an anonymous bind may be granted read-only access to part of the tree. The nature of your application and the capabilities of your clients dictate how you allow or disallow various authentication methods.

If you wish to maintain a higher level of availability, then enable `gentlehup`. With this command enabled, `slapd` stops listening on the network when it receives a `SIGHUP` signal, but it doesn't drop any open connections. A new instance of `slapd` can then be started, usually with an updated configuration.

To get more verbose logging, adjust the value of `loglevel`. This command accepts an integer, multiple integers, or a series of keywords, which enable logging for a particular function. Consult the slapd.conf manpage for the full list of keywords and values. For example, connection tracing has a value of 8 and a keyword of `conns`, and synchronization has a value of 4096 and a keyword of `sync`. To enable logging of these two items `logging 5004`, `logging 8 4096`, or `logging conns sync` will achieve the same result.

If you compiled OpenLDAP from source, you may have enabled some modules. Alternatively, you may have downloaded extra modules from your package manager, such as the `openldap-server-sql` package, which includes the SQL backend module. The `modulepath` and `moduleload` options are used to load dynamic modules into `slapd`. `modulepath` specifies the directory (or list of directories) that contains the shared libraries, and each instance of `moduleload` specifies a module to load. It's not necessary to specify the module's version number or extension, because `slapd` looks for a shared library. For example, for a library called `back_sql-2.3.so.0.2.18`, use `moduleload back_sql`. Alternatively, `moduleload` can be given the full path (without the version and extension) to the library, such as `moduleload /usr/share/openldap/back_sql`.

Some scripts expect the process id of a process to be held in a certain file. `pidfile` tells `slapd` where to write its process id.

Schema parameters

A handful of commands let you add schema items to your tree, either by including a schema file or by defining the object in slapd.conf. Recall from the previous tutorial that the schema provides the attributes and object classes that can be used by your LDAP tree.

To add a new schema file to your server, use the `include` command followed by the full path to the schema file (usually found in /etc/openldap/schema). If one schema makes reference to another (such as `inetOrgPerson` inheriting from `organizationalPerson`), you need to include all the necessary files in the proper order, with the base objects included first. OpenLDAP parses each schema file as it's included, so order of inclusion is important.

You can add new schema items directly through slapd.conf with the `attributetype` and `objectclass` commands for attributes and object classes, respectively. This is the same as putting the information in a schema file and including it with the `include` command. Similarly, you can define object identifiers (OIDs) with `objectidentifier`.

Backends and databases

Backends and databases are two separate but closely related concepts. A database represents part of a tree, such as `dc=ertw,dc=com`. A backend describes the method by which `slapd` retrieves the data. (The `dc=ertw,dc=com` tree has been the primary example in this series.)

In many cases, the backend is a file on disk (in some format; more on this later); or it can be a method to get data from another source, from a SQL database, to DNS, and even through a script. Each database is handled by one backend, and the same backend type can be used by multiple databases.

As noted earlier, slapd.conf starts with global directives. Backend mode then starts at the first instance of the `backend` directive. All directives in this backed mode apply to the particular backend being configured. Any options that were set globally

apply to the backend, unless they're overridden at the backend level. Similarly, you configure databases with the `database` keyword. A database is tied to a backend type, which inherits any global or backend level configurations. You can override any options at the database level, too.

OpenLDAP splits the backends into three types:

1. Those that store data:
   - bdb—Uses the Berkeley database engine (such as Sleepycat, now owned by Oracle)
   - hdb—An improvement on back-ldb, which adds some indexing improvements
2. Those that proxy data:
   - ldap—Proxies another LDAP server
   - meta—Proxies several LDAP servers for different parts of the tree
   - sql—Returns data from a SQL database
3. Those that generate data:
   - dnssrv—Returns LDAP referrals based on data in DNS SRV records
   - monitor—Returns statistics from the LDAP server
   - null—A testing module; returns nothing
   - passwd—Returns data from the password file
   - perl—Returns data generated from a Perl script
   - shell—Returns data generated from a shell script

Configuration options are specific to each backend, and can be found in the relevant manpage (such as `slapd-bdb` for the bdb backend).

Databases represent the tree and its data. The `dc=ertw,dc=com` tree is an example of a database. All data under this DN would be stored in a similar fashion if it were part of the same database. It's also possible to have `ou=people,dc=ertw,dc=com` in one database, with anything else under `dc=ertw,dc=com` in another. Finally, an LDAP server can serve more than one tree, such as `dc=ertw,dc=com` and `dc=lpi,dc=org`. Each database has its own way of handling the request by way of its own backend.

Specify `database` followed by the database type to start database configuration mode. The commonly used form is the Berkeley database, so `database bdb` creates a BDB database. The next command you need is `suffix`, which specifies the root of the tree the database is serving.

`rootdn` and `rootpw` allow you to specify a user with all privileges (a *root user*) for

the database. This user isn't even subject to access controls. The `rootdn` should be within the specified suffix and may or may not have a password. If a `rootpw` is specified, this is used. Otherwise, the behavior is to look for the `rootdn`'s record in the tree and authenticate against the `userPassword` attribute. If no root user is specified, then all users are subject to the access controls configured.

If you specify `lastmod on`, OpenLDAP keeps several hidden attributes (called *operational attributes*), such as the name of the person who created the record and when it was modified. Some of these attributes are required for replication to work, so it's smart to leave `lastmod` enabled (which is the default). These operational attributes aren't shown to clients unless specifically requested.

You can further restrict what can be done to the database through the `restrict` command. This command takes parameters corresponding to LDAP operations, such as `add`, `bind`, `compare`, `delete`, `rename`, and `search`. To block users from deleting nodes in the tree, use `restrict delete`. If the tree contains users, but for some reason you don't want them to be able to bind to the tree, use `restrict bind`. Additionally, `read` and `write` are available to block any reading and writing to the tree, respectively, rather than having to spell out all the relevant operations. Alternatively, you can use the command `readonly` to make the database read only.

Different parts of the same tree can be handled by different databases. If properly configured, OpenLDAP glues all the parts together. The database containing the other is called the *superior database*; the database being contained is the *subordinate database*. First, define the subordinate database and add the `subordinate` command on a line of its own. Then, define the superior database. With this configuration, OpenLDAP can treat multiple databases as one, with some data stored locally and some pulled from other sources (a special case of this is when all the data is on remote LDAP servers, which is where a metadirectory is used). Note that if you define the superior database before the subordinate database, you'll get errors that you're trying to redefine part of your tree. Listing 5 shows the `dc=ertw,dc=com` tree split into a superior and a subordinate database.

Listing 5. Configuration for a subordinate and superior database

```
# Subordinate
database bdb
suffix "ou=people,dc=ertw, dc=com"
rootdn "cn=Sean Walberg,ou=people,dc=ertw,dc=com"
rootpw mysecret
directory       /var/db/openldap/ertw-com-people
subordinate

# Superior
database bdb
suffix "dc=ertw, dc=com"
rootdn "cn=Sean Walberg,dc=ertw,dc=com"
rootpw mysecret
directory       /var/db/openldap/ertw-com
```

Also note that two `rootdn`s are configured. If you want to define a password, the `rootdn` must fall within the database. To build the tree, the second root account must be used to define the `dc=ertw,dc=com` entry, and the first root account

defines the people organizational unit (OU) and any objects underneath it. Once users have been added, you can authenticate as a different user in order to get access to the whole tree.

If you're using the bdb backend, you also need to use the `directory` command to specify where the database files are stored. Each database instance needs a separate directory.

Setting up a new database is fairly simple, because there are only a few commands to worry about. Much of the complexity comes in when you try to tune the backend, which is the subject of the next tutorial in this 301 series.

Overlays

Overlays are an extension of the database. If you want to add a feature to a database, you can often add it as an overlay rather than forking the database code. For example, if you want all writes to be logged to a file, you can attach the `auditlog` overlay to the relevant database.

Overlays operate as a stack. After configuring the database, you specify one or more databases. Then, define each overlay with the `overlay` command, followed by the name of the overlay. Each overlay has its own configuration parameters.

If you've configured multiple overlays, they're run in the reverse order that you define them. The database is accessed only after all the overlays have run. After the database returns the data, the overlays are run again in the same order before `slapd` returns the data to the client.

At each step, an overlay can perform an action such as logging, it can modify the request or response, or it can stop processing.

Section 3. Developing for LDAP with Perl/C++

This section covers material for topic 302.2 for the Senior Level Linux Professional (LPIC-3) exam 301. This topic has a weight of 1.

In this section, learn how to:

- Use Perl's `Net::LDAP` module
- Write Perl scripts to bind, search, and modify directories
- Develop in C/C++

Although OpenLDAP includes command-line clients, it's often helpful to use LDAP information in your own scripts. Perl is a popular language for scripting. Perl has a module called `Net::LDAP` that is used to connect to and use an LDAP server.

Getting started

`Net::LDAP` doesn't ship with Perl, but your distribution may include it as a package. See "Installing from packages" for more information on searching for and installing packages.

If your distribution doesn't have the `Net::LDAP` package, then you can download it from the Comprehensive Perl Archive Network (CPAN). As root, run `perl -MCPAN -e "install Net::LDAP"`, which downloads and installs `Net::LDAP` and any dependencies.

Using Net::LDAP

Using `Net::LDAP` is fairly simple:

1. Create a new `Net::LDAP` object.
2. Bind to the desired server.
3. Perform your LDAP operations.

Create a new object

In typical Perl fashion, you must create an instance of the `Net::LDAP` module through the `new` method. All further operations will be on this instance. `new` requires, at a minimum, the name of the server you want to connect to. For example:

```
my $ldap = Net::LDAP->new('localhost') or die "$@";
```

Here, a new `Net::LDAP` object is created with the `new` method and is passed the string `localhost`. The result is assigned to the `$ldap` variable. If the function fails, the program exits and prints an error message describing the problem. $@ is a Perl internal variable that contains the status of the last operation.

You can proceed to perform LDAP operations with the new `Net::LDAP` object. Each function returns a `Net::LDAP::Message` object that contains the status of the operation, any error messages, and any data returned from the server.

Binding to the tree

The first operation you should do is to log in or *bind* to the tree. Listing 6 shows a bind operation and associated error checking.

Listing 6. Perl code to bind to the tree

```
my $message = $ldap->bind(
        "cn=Sean Walberg,ou=people,dc=ertw,dc=com",
        password=>"test" );
```

```
if ($message->code() != 0) {
        die $message->error();
}
```

Listing 6 starts by calling the `bind` method of the previously created object. The first parameter to the function is the DN you're binding as. If you don't specify a DN, you bind anonymously. Further parameters are in the format of `key=>value`; the one you'll use most often is the password.

Each `Net::LDAP` method returns a `Net::LDAP::Message` object, which has the results of the function. The error code is retrieved through the `code` method. A code of 0 means success, so the code in Listing 6 exits the program with the error message if the result isn't 0. Note that the error is retrieved from `$message->error` rather than $@, like the earlier example. This is because the error isn't a Perl error; it's internal to `Net::LDAP`.

Once the bind is successful, you can do anything you want, subject to the server's access controls. To log out, call the `unbind` method.

Searching the tree

Searching is done through the `search` method. Like the `bind` method, you must pass some parameters and check the result of your query. However, the returned object now contains your data, so this must be parsed. With the `search` operation, the result is a `Net::LDAP::Search` object, which inherits all the methods from `Net::LDAP::Message` (such as `code` and `error`) and adds methods to help you parse the data. Listing 7 shows a search of the tree.

Listing 7. Searching the tree with search

```
$message = $ldap->search(base => "dc=ertw,dc=com", filter=> "(objectClass=*)");
if ($message->code() != 0) {
        print $message->error();
} else {
        foreach my $entry ($message->entries()) {
                print $entry->dn() . ": ";
                print join ", ", $entry->get_value("objectClass");
                print "\n";
        }
}
```

Listing 7 begins by calling the `search` method, passing two parameters: the base and a filter. The base tells the server where in the tree to begin searching. A complementary option, `scope`, tells the server how far to search:

- **base** —Only the base object
- **one** —Only the children of the base object (and not the base object itself)
- **sub** —The base object and all its children (the default)

The filter is a string describing the objects you're interested in. You can search on attributes and perform complex AND/OR queries. `objectClass=*` returns any

object.

The result of the search is checked, and an error is printed if a problem happened. Because the script could still recover from an error, it just prints the error and continues, rather than exiting.

The `entries` function returns an array of `Net::LDAP::Entry` objects, each with a single result. First the entry's DN is printed, and then all the object classes. If you'd rather have a text version of the whole record, the `dump` method prints the entire entry in text format.

Adding a new entry

You add an entry to the tree through the `add` method. You must pass the function the DN of the entry you wish to add, along with the attributes. The attributes are an array of `key =>` value pairs. The value can also be an array in the case of multiple instances of the same attribute. Listing 8 shows an entry being added to the tree.

Listing 8. Adding an entry using Net::LDAP

```
$message  = $ldap->add(
        "cn=Fred Flintstone,ou=people,dc=ertw,dc=com",
        attr => [
                cn   => "Fred Flintstone",
                sn   => "Flintstone",
                objectclass => [ "organizationalPerson",
                        "inetOrgPerson" ],
        ]
);

if ($message->code() != 0) {
        print $message->error();
}
```

The first parameter to `add` is either the DN or a `Net::LDAP::Entry` object. If the DN is passed, you must pass an arrayref through the `attr` method. Even though the `key =>` value format is used as in a hashref, `Net::LDAP` is expecting an arrayref, so be careful!

More about Net::LDAP

`Net::LDAP` provides an interface to all the LDAP functions, such as `compare`, `delete`, and `moddn`. They're all used similarly to the previous examples and are fully documented in the `Net::LDAP` manpage.

All the examples shown operate in blocking mode, which means the function returns after the response has been received from the server. You can also operate in asynchronous mode, which involves giving a callback function that is called as packets are received.

By using `Net::LDAP`, you can use the data stored in your LDAP tree from within your scripts. Perl is already used in a wide variety of software, so the opportunities for integration are unlimited.

Developing in C/C++

Using the C libraries is more involved than the Perl libraries. The ldap(3) manpage contains a detailed description of how to use the library, and has pointers to the other manpages describing each function. To use the LDAP C libraries, your code must first include the ldap.h include file, such as with `#include <ldap.h>`. Your object files must then be linked with libldap using the `-lldap` option to the linker.

Section 4. Summary

In this tutorial, you learned about installing and configuring the OpenLDAP stand-alone server. When configuring `slapd`, use the slapd.conf file. You must take care to keep your global options at the top of the file and then progress to backend and database configurations, because `slapd` is dependent on the order of the directives. When in doubt, consult the slapd.conf manpage.

Perl code can make use of an LDAP server through the `Net::LDAP` module. First you create an object, and then you call methods of the object that correspond with the LDAP operation you want. Generally, you first `bind` and then perform your queries. It's important to check the results of your functions through the `code` and `error` functions.

Resources

Learn

- Review the previous tutorial in this 301 series, "LPI exam 301 prep, Topic 301: Concepts, architecture, and design" (developerWorks, October 2007).
- Take the developerWorks tutorial "Linux Installation and Package Management" (developerWorks, September 2005) to brush up on your package management commands.
- Review the entire LPI exam prep tutorial series on developerWorks to learn Linux fundamentals and prepare for system administrator certification.
- At the LPIC Program, find task lists, sample questions, and detailed objectives for the three levels of the Linux Professional Institute's Linux system administration certification.
- Find answers to What is a backend? and What is a database? in the OpenLDAP FAQ.
- Learn more about building overlays in the developer's documentation. Overlays are a complex but powerful concept.
- Consult the *OpenLDAP Administrator's Guide* if the manpages for slapd.conf or your particular backend don't help. You might also find the OpenLDAP FAQ to be helpful.
- The online book *LDAP for Rocket Scientists* is excellent, despite being a work in progress.
- The Perl-LDAP page has lots of documentation and advice on using `Net::LDAP`.
- In the developerWorks Linux zone, find more resources for Linux developers, and scan our most popular articles and tutorials.
- See all Linux tips and Linux tutorials on developerWorks.
- Stay current with developerWorks technical events and Webcasts.

Get products and technologies

- Download OpenLDAP.
- The IBM Tivoli Directory Server is a competing LDAP server that integrates well with other IBM products.
- phpLDAPadmin is a Web-based LDAP administration tool. If the GUI is more your style, Luma is a good one to look at.
- Order the SEK for Linux, a two-DVD set containing the latest IBM trial software for Linux from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.
- With IBM trial software, available for download directly from developerWorks, build your next development project on Linux.

Discuss

- Get involved in the developerWorks community through blogs, forums, podcasts, and community topics in our new developerWorks spaces.

About the author

Sean Walberg
Sean Walberg has been working with Linux and UNIX since 1994 in academic, corporate, and Internet service provider environments. He has written extensively about systems administration over the past several years.

Trademarks

DB2, Lotus, Rational, Tivoli, and WebSphere are trademarks of IBM Corporation in the United States, other countries, or both.
Linux is a trademark of Linus Torvalds in the United States, other countries, or both.
UNIX is a registered trademark of The Open Group in the United States and other countries.
Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.